2015 IEEE 39th Annual International Computers, Software & Applications Conference

On the Validity of the IT-CISQ Quality Model for Automatic Measurement of Maintainability

Reinhold Plösch and Severin Schürz
Dept. of Business Informatics – Software Engineering
Johannes Kepler University
Linz, Austria
reinhold.ploesch@jku.at, severin.schuerz@gmx.at

Christian Körner
Corporate Technology
Siemens AG
Munich, Germany
christian.koerner@siemens.com

***Abstract*— IT-CISQ (Consortium for IT Software Quality) standard that was initiated and is driven by an OMG (Object Management Group) consortium claims to provide operational measures for a number of quality criteria of the ISO / IEC 25010 quality standard. In order to show the applicability of the IT-CISQ standard, we built a tool that provides automatic measures for the maintainability related measures of the IT-CISQ standard. We evaluated five Java open source projects with our IT-CISQ quality model and compared these results with the result of an expert-centered evaluation of the maintainability of these projects. The study shows that the IT-CISQ quality model for maintainability is well suited as a predictor for the quality criterion maintainability. Compared to our more comprehensive EMISQ (Expert Centered Method for Internal Software Quality) quality model (165 EMISQ measures vs. 18 IT-CISQ measures), the IT-CISQ quality model yields a similar order of projects. However, it is of course less suitable to give good guidance about improving aspects, as our more comprehensive EMISQ quality model includes considerably more maintainability related aspects of software. Furthermore, the tooling provided by us is innovative and could directly be used in practice by itself.**

Keywords-software quality, ISO / IEC 25010, IT-CISQ quality model, maintainability

I. INTRODUCTION

Despite substantial efforts in both research and practice, software quality continues to be a controversial and insufficiently understood issue, and the quality of software products is often dissatisfactory. The economic impact is enormous and includes not only spectacular failures of software but also increased maintenance costs, high resource consumption, long test cycles, and delayed features.

Software quality models try to tackle the issues described above by providing a systematic approach for modeling quality requirements. Some of the models additionally support analyzing and monitoring quality as well as mechanisms for directing quality improvement measures [1].

In practice, however, typically a gap remains between the abstract quality characteristics described in quality models (e.g., ISO / IEC 25010) [2] and concrete measurements and assessments [3]. Missing concrete measurements makes it difficult to explain the importance of quality problems to developers or sponsors and to quantify the potential of quality improvements. A similar gap exists for quality assessment methods. Effective quality management requires an objective method for assessing the overall quality of a software product based on measured properties. Existing quality models often provide procedures that are too abstract to be operational (e.g., ISO/IEC 25040) [4].

In previous work ([5], [6]) we developed operational quality models, i.e., quality models that provide concrete measurements for typical software quality attributes. Additionally, our EMISQ (Expert Centered Method for Internal Software Quality) quality model [6] provides a benchmarking-based assessment method that was validated in several studies (see [7] and [8]), indicating that the results achieved with our assessment approach have high external validity. Our Quamoco model [5], e.g., provides an even more elaborate but costly assessment method that also showed a high external validity of the results in several experiments.

The IT-CISQ (Consortium for IT Software Quality) quality model proposed by the OMG (Object Management Group) in 2012 [9] claims to close the gap between abstract quality characteristics as suggested by the ISO/IEC 25010 quality model and concrete measurements for the quality attributes Maintainability, Reliability, Performance efficiency, and Security. Providing this necessary operationalization for the ISO/IEC 25010 standard pushes the IT-CISQ model in the spotlight for practitioners – either to have at least something to hold on, or because of pressure imposed on organizations to follow this standard. It therefore is of vital importance for industry to find out whether the proposed standard is yet ready to be used.

The research objective of this paper is to validate the IT-CISQ quality model for the ISO/IEC 25010 quality characteristic maintainability, i.e., to validate, whether the measures defined in the IT-CISQ quality model for maintainability can be reasonably used for assessing the maintainability of software products. To achieve this objective it was necessary to implement measures for maintainability as specified by the IT-CISQ quality model. We used our benchmarking-based approach for quality assessment [11] in order to judge whether the IT-CISQ quality model can properly measure the maintainability of software. Although the IT-CISQ quality model provides a rough evaluation method, we used our benchmarking-based evaluation method that was also applied in our previous studies in order to be able to concentrate on the goal of this

0730-3157/15 $31.00 © 2015 IEEE
DOI 10.1109/COMPSAC.2015.47

IEEE computer society

validation study. With two different evaluation methods in place, it would be difficult or even impossible to separate the impact of the evaluation method from the impact of the IT-CISQ measures for maintainability on the results.

The paper has three major contributions: First, we developed a tool that allows automatic measurement of the maintainability-related measurements of the IT-CISQ quality model. Second, we adapted our benchmarking-based approach for quality assessment to the IT-CISQ quality model and therefore provide an assessment approach for the IT-CISQ model. Third, we performed a validation of the operational IT-CISQ quality model for maintainability with real software systems. This validation showed the correspondence of the assessment results with expert opinions.

The remainder of this article is structured as follows: Section II presents an overview of the IT-CISQ quality model and a more detailed description of the maintainability-related parts of the model. Section III presents our approach and tool-support for operationalizing the measurements specified by the IT-CISQ quality model for maintainability. Section IV describes the application and adoption of our benchmarking-based approach to the IT-CISQ model. Section V describes the validation of the IT-CISQ quality model for maintainability. Finally, Section VI concludes the article and outlines future work.

II. The IT-CISQ Quality Model

The goal of the IT-CISQ quality standard [9] is to provide specifications for measuring software quality. The IT-CISQ standard is based on the ISO/IEC 25010 standard for software quality, meaning that IT-CISQ does not provide a model for specifying software quality but instead relies on the definitions provided by ISO/IEC 25010. The IT-CISQ model enhances the ISO/IEC 25010 standard by providing a clear specification as to how to measure the defined quality attributes. It is claimed that this facilitates the definition of Service Level Agreements, the quantification of quality of business-critical applications, and supports the definition of benchmarks.

The standard is comprised of five sections. Section 1 introduces basic terms in the context of measurement. Section 2 defines terms with a heavy reuse of definitions from the ISO/IEC 25000 standards family. Section 3 defines requirements for implementing the standard. Section 4 is the core part of the standard and defines concrete measures for evaluating selected software quality attributes of the ISO/IEC 25010 standard. The current version 2.1 of the standard covers the quality attributes Reliability, Performance efficiency, Security and Maintainability.

Section 5 of the standard defines a rough calculation model for the aggregation of individual measures to overall indicators for a quality attribute. The IT-CISQ model does not refer to the next (finer) level of quality attributes. For example, the ISO/IEC quality attribute reliability is directly measured by the IT-CISQ model, and more differentiated measurement specifications for the sub-quality attributes of reliability (e.g., *maturity* or *availability*) are not provided.

The description of the operationalization of a quality attribute always follows the same structure. A number of *issues* are refined into *quality rules* that are operationalized by *quality measure elements*.

TABLE 1: ISSUES, QUALITY RULES AND QUALITY MEASURE ELEMENTS

Issue	Quality Rule	Quality Measure Element
Issue 1: In a layered architecture, functions should strictly be allocated to layers and maintain a strict hierarchy of calling between layers (utility layers excepted)	**Rule 1:** Functions only communicate (exchange data) with functions belonging to an adjacent layer. Functions do not directly exchange data with functions that are not in adjacent layer (no layer skipping / bridging)	**Measure 1:** # functions that span layers **Measure 2:** # of layer-skipping calls
	Rule 2: Avoid too many horizontal layers	**Measure 3:** # of layers (threshold 4 <= 8)

Table 1 shows an example of an issue that is related to layered architectures (therefore substantiating the ISO/IEC 25010 quality attribute maintainability) and refined into two rules and three measures.

Quality measure elements are typically formulated as rules and the underlying revealed metric is the number of elements in the source code that violate this rule. Subsequently we will use the general term measure, as the IT-CISQ standard also heavily relies on this term.

III. Measuring Maintainability

In this work we focus on measuring maintainability as defined by the IT-CISQ quality model. We chose maintainability since it is (1) an important quality attribute and (2) we have data from previous automatic assessments as well as from expert-centered assessments available that can be used to validate the IT-CISQ quality model.

In the previous section we presented the overall structure of the IT-CISQ model, i.e. from issues to quality rules and finally to quality measure elements (*measures* for short). For the purpose of evaluation, we only need to concentrate on the section of quality measures defined for maintainability by the IT-CISQ standard. Table 2 lists the measures as defined by the IT-CISQ. The rule number given in the first row will be used in the remainder of the paper.

As a first step for automating measurement, we analyzed the definitions and specifications given in the standard and tried to identify issues related to these specifications. The result of this analysis shows that these specifications are not exact enough for some measures:

M1, M2, M3: The standard makes the implicit assumption that each function is part of a layer. Furthermore the specification leaves open in which cases a function spans a layer (e.g., just when other functions are called, or also when variables or constants are used from other layers). Additionally, operationalization is difficult, since in order for each project to be analyzed the layers have to be specified manually.

M8: This measure is not generally applicable, as some programming languages (e.g., Java, C#) do not offer multiple implementation inheritance.

M11: Even though there are different approaches available in the literature there are no details given for calculating fan-out.

M12: Even though there are different approaches available in the literature there are no details given for calculating coupling between objects.

M18: The language specific thresholds remain unspecified in the standard.

M19: Although the specification is clear for this measure, operationalization is difficult, as for each project to be analyzed it has to be manually specified which functions, methods, classes, or packages contain data or file operations.

Table 3 lists which assumptions we took in order to be able to provide automatic measurement for the specified measures of the IT-CISQ standard.

TABLE 2: IT-CISQ MEASURES FOR MAINTAINABILITY

ID	Short Description of Measure
M1	# functions that span layers
M2	# of layer-skipping calls
M3	# of layers (threshold 4 <= 8)
M4	# files that contain 100+ consecutive duplicate tokens
M5	# of unreachable functions
M6	# of classes with inheritance levels >= 7
M7	# of classes with >= 10 children
M8	# of instances of multiple inheritance of concrete implementation classes
M9	# of methods that are directly using fields from other classes
M10	# of variables declared public
M11	# of functions that have a fan-out >= 10
M12	# of objects with coupling > 7
M13	# of cyclic calls between packages
M14	# of functions with > 2% commented out instructions
M15	# files > 1000 LOC
M16	# of instances of indexes modified within its loop
M17	# of GO TOs, CONTINUE, and BREAK outside the switch
M18	# of functions with cyclomatic complexity >= a language specific threshold
M19	# of methods with >= 7 data or file operations.
M20	# functions passing >= 7 parameters
M21	# of hard coded literals except (-1, 0, 1, 2, or literals initializing static or constant variables)

Figure 1. gives an overview of our tool architecture for the implementation of the necessary measures for IT-CISQ. *Understand* is a commercial tool[1] that parses source code (e.g., Java Source code) and stores the extracted information about types, methods, method parameters, dependencies between classes and packages, etc. in a database. Furthermore, the tool *Understand* provides a Perl-based API for accessing this stored information.

TABLE 3: ASSUMPTIONS FOR RULE AUTOMATION

M1, M2, M3	We assume that every function or method is always part of a layer. The implementation of a function or method is allowed to call functions or methods from *direct* upper or lower layers, but not from layers that are far away. In order to automatically calculate this measure, it has to be assured (by configuration) that each function or method (or its more high-level constructs like classes or packages) is assigned to exactly one layer.
M8	No meaningful implementation possible for Java. For this reason M8 is not considered anymore.
M11	The fan-out is calculated by summing up the number of called functions (methods) and the number of member variables set.
M12	For the calculation of the object coupling we rely on the well-known metric *Coupling Between Object* (CBO) as specified in [13].
M18	As threshold value for the cyclomatic complexity, we assume a value of 10 as proposed by McCabe [14] and because of the widespread use of this threshold value.
M19	We allow explicit specification for each project which packages or classes contain data or file operations.

MUSE (**M**USE **U**nderstand **S**cripting **E**ngine) is a library developed by us. This Perl library provides basic functionality for accessing the *Understand* database, for dealing with threshold values, for writing result files (in Figure 1. denoted as *findings.xml*), as well as a set of helper functions to more easily access the information provided by *Understand* (e.g., getting the package of a class, getting the list of all public classes of a project, etc.).

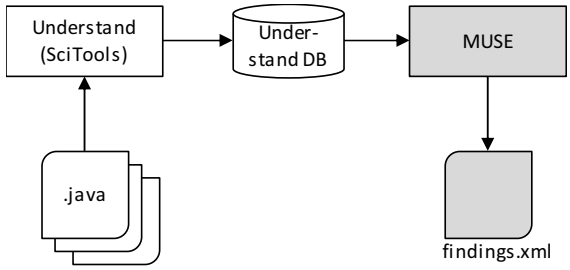

Figure 1. Tool architecture for implementing IT-CISQ measures

Furthermore, MUSE uses the module concept of Perl for grouping measures. These modules can be seen as small plugins. In case a module is stored in a specific directory of MUSE, it can be loaded and the measures defined in this module can be executed. This facilitates the configuration of the functionality of MUSE. For the implementation of the IT-CISQ measures, each of the 20 rules[2] is implemented as a separate Perl module in order to have maximum flexibility

[1] https://scitools.com/

[2] IT-CISQ specifies 21 measures for maintainability, but measure M8 is not applicable for Java projects (see Table 3)

for the validation. The implementation of the 20 IT-CISQ rules has ~2,000 LOC. Nevertheless, as mentioned before the implementation relies on some basic functionality provided by the MUSE framework. The MUSE-based tooling provided by us is by itself innovative and provides at least an efficient way to get the results for the IT-CISQ defined measures.

IV. Assessment Method For Maintainability

Benchmarking is a well-established concept in many business areas where (similar) objects (e.g., products, services, processes, organizations) are to be checked against each other for specific purposes, e.g., for the evaluation of the value of an object or to derive suggestions for improvement.

In the context of software quality, Simon et al. [10] present a method for expressing the quality of a software product by calculating a single grade named Code-Quality-Index (CQI). This method applies benchmarking for the calculation of the CQI by comparing metric values of a product with statistical data from a benchmark base. The benchmark base holds aggregated statistical data of more than 100 commercial software systems. However, the CQI calculation model suffers some conceptual, benchmarking-related, and technical deficiencies as investigated and described in [11].

Based on our experiences with the CQI, we developed an automatic assessment approach for software quality [12] that is based on the benchmarking concept. For this, in a first step we built up a benchmark database that stores the data of all reference projects included, i.e., projects that are used for comparison. This means that we store the results of the IT-CISQ quality measures for maintainability (see Section III). It is sufficient in this context just to count for each rule the number of violations per project of the benchmark base. Additionally, we store values of various size metrics for each project needed for normalization; examples of size metrics are lines of code, number of methods, or number of classes.

Normalization of a measurement result deals with making raw results comparable with other projects. As an example, having 10 duplicated code fragments in the source code might be tolerable in a project with 300k lines of code. But in a project with only 10k lines of code, 10 code duplications might be unacceptable. For this reason we use proper normalization to eliminate the inherent project contexts and make measurement values comparable with those of other projects.

Based on the benchmark database that can be used for several assessments, a benchmark suite needs to be generated in a second step. A benchmark suite usually is specially tailored to a specific application of benchmarking and consists of only those projects that are selected as reference projects, e.g., limited to projects of the embedded systems domain, which are written in the C programming language. In our context (more details will be given in Section V), the benchmarking suite consists of 26 open-source Java projects.

To create the benchmark suite, we calculate a value distribution for each measure on the basis of their normalized measurement values of the selected projects. These value distributions reflect commonly used statistical figures, e.g., quartiles, quintiles, deciles, or percentiles.

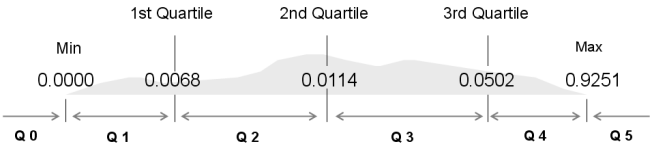

Figure 2. Quartile-based value distribution for one rule (example) [12]

A quartile-based distribution, as shown in Fig. 2, divides the values into four areas, Q1-area to Q4-area, where Q1-area and Q4-area are delimitated with the minimum and maximum value of the suite projects. If an investigated project is better than the benchmark minimum it is in the additional Q_{min}-area. Vice versa, this also applies to the maximum value.

In our original work [12] we calculate an average quality score for the project being evaluated. For this, we consider all rules/metrics and average out the number of rules/metrics falling into each quality area from Q_{min}-area to Q_{max}-area. In order to calculate the score, we assign a value for each Q-result; the better the area, the higher the number. For example, a Q-result of Q0 will get 5 points while a result of Q5 will get 0 points. Summing up the points for all rules and dividing this sum by the total number of selected rules produces the quality score.

In addition to the quality score, we also use a diagram that visualizes these facts. For that we draw a colored chart with bars showing the number of rules that have a particular Q-result. Figure 3. shows the diagram and the calculation of the quality score based on this data. The quality score is not an absolute value for making a quality statement, as the quality score depends on reference projects that have been used for calculating the benchmark suite.

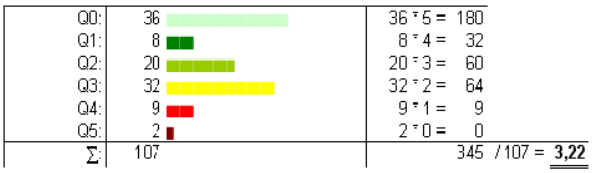

Figure 3. The benchmark diagram including the calculation of the quality score

For the validation of the maintainability measures of the IT-CISQ model we rely on our benchmarking-based approach for quality assessment, as this approach was also used in the *Linzer Softwareverkostung*[3] (LSV) [7]. The remaining part of this paper concentrates on the validation of the developed measures for the maintainability of software as defined by the IT-CISQ standard.

[3] *Softwareverkostung* (german) means software tasting in analogy to wine tasting

V. VALIDATION

A. Overview of the Validation

In order to validate the IT-CISQ measures for maintainability, we will compare the results of these measures with the results of the LSV, published in [7] and [15]. Besides the two publications, all raw data of the LSV are available to us. The major goal of the LSV was to validate whether our benchmarking-based assessment method (described in Section IV) obtains similar results (with respect to the quality of the software) to the human experts. During the LSV, five Java open source projects were assessed – (1) Log4j 1.2.15, (2) JabRef 2.3.1, (3) TVBrowser 2.2.5, (4) RSSOwl 1.2.4, and (5) Checkstyle 4.4. The size of each software project is between 13,000 and 66,000 logical lines of code.

Based on this data, we describe two validation studies in this article:

(1) Assessing the quality of the five projects of the LSV using our IT-CISQ measures and relying on our benchmarking-based assessment method. The goal of this validation experiment is to figure out whether our approach obtains similar results to the experts of the LSV.
(2) Comparing the results of the automatic assessment performed during the LSV (using a large number of measures contained in our EMISQ quality model [6]) with the small number of IT-CISQ measures. The goal of this validation experiment is to find out whether the small number of measures can be used for judging the maintainability of Java software.

B. Study 1 – IT-CISQ Measures and Human Experts

1) Study Design

During the LSV (*Linzer Softwareverkostung*), ten experts assessed the five aforementioned projects throughout one day. These experts from industry had at least 7 years of development experience with Java software in an industrial context. The time allocated to the experts for each project was aligned with the size of the project. The experts had two major tasks: (1) Assessment of the quality criteria *Analyzability*, *Craftsmanship*, *Reliability*, and *Structuredness* for each project, with grades between 1 (very good) and 5 (very bad) applicable to each criterion; (2) Ranking of all five projects according to the overall quality.

The overall quality and ranking was derived by the experts based on individual judgements without explicitly specifying an evaluation method or explicitly weighing the quality criteria. Table 4 gives the results of this overall expert ranking.

There is a high correlation between the individual ratings of the experts. The average Spearman correlation coefficient is 0.6618, which is significant. Furthermore, if we consider the ranking of the projects and calculate Kendall's W concordance coefficient, we get a value of 0.6635, meaning that there is a strong concordance of the rankings of the individual experts with a significance level of $\alpha = 0.01$. We applied different variants for aggregating the individual rankings of the experts to one overall ranking that represents the overall combined view of all experts.

TABLE 4: EXPERT RANKING OF LSV – OVERALL RANKING (EO)

	Log4J	JabRef	TV-Brow.	RSSOwl	Checkstyle
Median	2	5	4	2	1
Average	3	5	4	2	1
Majority	2	5	4	3	1
Condorcet	3	5	4	2	1

Table 4 gives expert rankings using median values, arithmetic mean (average), majority, and Condorcet [16], the latter of which is a method that directly calculates the ranking. There are different methods available for calculating the Condorcet-based ranking; in this work we used the algorithm defined in [17].

As the quality criterion reliability is not associated with maintainability, we also recalculated the overall rating of the five projects based on the individual ratings for analyzability, craftsmanship, and structuredness, i.e., omitting reliability. The result is shown in Table 5.

TABLE 5: EXPERT RANKING OF LSV BASED ON QUALITY CRITERIA AGGREGATION (EC)

	Log4J	JabRef	TV-Brow.	RSSOwl	Checkstyle
Median	3	5	4	1	1
Average	3	5	4	2	1
Majority	1	5	4	1	1
Condorcet	3	5	4	2	1

What we see from Table 4 and Table 5 is that, regardless of the aggregation method used, *Checkstyle* is the best project and *JabRef* is the worst project. The rank of the *TV-Browser* project is also always 4. Only *Log4J* and *RSSOwl* have slightly different ranks depending on the aggregation method used.

Table 4 and Table 5 therefore represent the results of the overall ranking of the experts. Calculating these ranking values was a preparatory task. The core of the study is of course to apply the maintainability measures of IT-CISQ to the five projects of the LSV using our benchmarking approach. To prepare our benchmarking approach, we applied our IT-CISQ measures to 26 open-source Java projects in order to get valid benchmarking distributions for all IT-CISQ measures (see Section III). The IT-CISQ measures M1, M2, M3, and M19 need project specific configuration. This is difficult to achieve, as we would have to provide these project specific configurations for the 26 projects of the benchmark base and for the five projects of the LSV. This is not possible from the point of view of effort. Additionally, we do not know the layering architecture of the projects that is needed for M1, M2, and M3. We therefore calculated two variants of the benchmarking distribution:

- *Variant A* does not consider the IT-CISQ measures M1, M2, M3, and M19.

- *Variant B* considers all IT-CISQ measures. For the measures M1, M2, and M3 it is necessary to provide architectural layers. For this purpose we used the tool *Classycle* [18] that automatically provides a layering. This is of course not optimal, as the actual layering of projects will always be different from the *Classycle* results. For M19 we defined a set of packages and classes of the Java JDK that contain file operations – other libraries or packages (e.g., db4o [19] or *Apache Commons IO* [20]) possibly used by the projects are not considered.

Given the benchmark distributions (*Variant A* and *Variant B*) we assessed the five projects of the LSV and compared the results of the automatic assessment with the expert ratings previously presented in this section.

2) Study Results and Discussion

By using *Variant A* of the IT-CISQ measures, the benchmarking-based approach for the five projects yields the ranking as depicted in Table 6.

TABLE 6: IT-CISQ RANKING VARIANT A

	Log4J	JabRef	TV-Brow.	RSSOwl	Checkstyle
Avg4 Var. A	3.33	1	2	2.33	4.33
Rank Var. A	2	5	4	3	1

The row *Avg4 Var. A* is the direct calculation result using the benchmark calculation method as described in Section III. For this benchmark calculation we used *Variant A* of the benchmark distribution, i.e., without measures M1, M2, M3, and M19. We see from the results that the IT-CISQ measures *Variant A* also judge *Checkstyle* as the best project and *JabRef* as the worst project; TV-Browser gets rank 4, which is also in concordance with the results of the experts (see Table 4 and Table 5).

Using *Variant B* of the IT-CISQ measures, the results and rankings are given in Table 7. The row *Avg4 Var. B* is the direct calculation result using the benchmark calculation method as described in Section III. For this benchmark calculation we used *Variant B* of the benchmark distribution, i.e., with our default configuration for measuring M1, M2, M3, and M19. We see from the results that the IT-CISQ measures *Variant B* also have *Checkstyle* as the best project and *JabRef* as the worst project. There is more variation with the other projects – e.g. *TV-Browser*, which always got rank 4 in all expert evaluations (regardless of the aggregation method), gets rank 2 using *Variant B* of the IT-CISQ measures.

TABLE 7: IT-CISQ RANKING VARIANT B

	Log4J	JabRef	TV-Brow.	RSSOwl	Checkstyle
Avg4 Var. B	2.72	1.61	2.72	2.5	3.33
Rank Var. B	2	5	2	4	1

For a systematic analysis of the correlations, we provide the results in Table 7. We calculated the correlation coefficients for four combinations:

(1) *EO-CA:* Expert centered ranking based on overall quality ranking (EO) (see Table 4) correlated with IT-CISQ measures *Variant A* (CA) (see Table 6).
(2) *EC-CA:* Expert centered ranking based on quality criteria aggregation (EC) (see Table 5) correlated with IT-CISQ measures *Variant A* (CA) (see Table 6).
(3) *EO-CB:* Expert centered ranking based on overall quality ranking (EO) (see Table 4) correlated with IT-CISQ measures *Variant B* (CB) (see Table 7).
(4) *EC-CB:* Expert centered ranking based on quality criteria aggregation (EC) (see Table 5) correlated with IT-CISQ measures *Variant A* (CA) (see Table 7).

TABLE 8: CORRELATION COEFFICENTS EXPERT CENTERED RANKING VS. IT-CISQ MEASURE-BASED RANKING

	EO-CA	EC-CA	EO-CB	EC-CB
Median	0.962**	0.795	0.255	0.495
Average	0.9*	0.9*	0.308	0.308
Majority	1**	0.892*	0.128	0.392
Condorcet	0.9*	0.962**	0.308	0.255

Correlation coefficients with one star denote a significant correlation assuming a level of significance of $\alpha = 0.05$. Entries with two stars are even significant at a level of significance of $\alpha = 0.01$. As the sample size is small (five projects), the correlation coefficient has to have a value of 0.878 ($\alpha = 0.05$) and 0.959 ($\alpha = 0.01$) to indicate a significant correlation.

What we see is that there is a significant correlation between the results of the expert-centered assessments and the IT-CISQ measures when using *Variant A*. On the other hand, there is no significant correlation at all when using *Variant B* of the IT-CISQ measures. One explanation for this is of course that our generic configuration for the IT-CISQ measures M1, M2, M3, and M19 does not fit. For measures M1, M2, and M3 we rely on the layers found by the tool *Classycle* (see above). A random check of the results provided by *Classycle* shows, e.g., for some projects no layering at all and for other projects (e.g., the Spring framework, that is part of the benchmarking base) a layering depth of 17, which also does not seem to be correct.

Finishing the discussion of the study results and therefore answering the first question of the validation section, we can say that *Variant A* of the IT-CISQ measures calculates the same ranking of Java projects as experts do for the quality criterion maintainability. The results are significant.

3) Threats to Validity

One threat to the internal validity of the study is the fact that the LSV did not exclusively concentrate on maintainability, but also on the quality criterion reliability. We tackled this problem by recalculating the expert rankings based on the detailed data for all maintainability related

quality criteria, i.e., excluding the data for reliability. Furthermore, there is no significant difference in the ranking (see Table 4 and Table 5).

The ranking of a small number of projects (in our case five projects) is a threat to internal validity. Nevertheless, we defined high minimum correlation coefficients to accept a correlation as significant (e.g., 0.878 for $\alpha = 0.05$) and therefore address this internal threat explicitly.

The study shows that *Variant A* of the IT-CISQ measures (excluding M1, M2, M3, and M19) can be used to judge the quality (focused on maintainability) of Java projects. As we were not able to positively validate the results of the entire set of IT-CISQ measures (i.e., *Variant B*), we could only partly show that the IT-CISQ measures for maintainability are suitable to predict the maintenance quality of Java projects. This is an external threat to validity.

The second external threat to validity is that our conclusions for this first study rely on one expert-based evaluation of maintenance quality only, namely our LSV. The LSV itself is solid (see [7] and [15]), as the number of experts (ten) is high enough, the experience of the experts is sufficient (7+ years of Java development experience), and there was sufficient time (one day for each expert) to assess the maintainability of the five projects.

C. Study 2 – IT-CISQ measures and EMISQ quality model

1) Study Design

In the context of the LSV, we also used our EMISQ quality model [6] to automatically determine the internal quality of the five projects of the LSV. In the original experiment the EMISQ quality model was also comprised of measures beyond maintainability, i.e., the model also contained measures for the quality criteria reliability and efficiency. In this study we want to find out whether the IT-CISQ measures are good predictors for the internal quality compared to the more comprehensive measure-set provided by our EMISQ quality model.

In order to be able to answer this question, we reworked our EMISQ quality model and excluded all measures that are not related to the quality criterion maintainability.

TABLE 9: NUMBER OF MEASURES (GROUPED BY TOOL) ORIGINALLY USED IN THE LSV AND USED IN THIS STUDY

	Original No. of Rules	*No. Of Rules Used in This Study*
FindBugs	88	28
PMD	130	87
SISSy	50	50
All Tools	268	165

Table 9 gives an overview of the number of rules we originally used (left column) and that were used for this study (right column). There is a substantial difference in the number of distinct rules. While the EMISQ quality model for maintainability comprises 165 measures (see Table 9), the Variant A of our IT-CISQ model comprises only 16 rules; roughly, a tenth of the EMISQ quality model.

Following our benchmarking approach, we first applied the newly adapted EMISQ quality model to our 26 projects of the benchmark base, calculated the benchmark distributions, and afterwards applied the adapted EMISQ quality model to the five projects of the LSV.
Table 10 gives the results.

TABLE 10: BENCHMARKING RESULTS OBTAINED BY APPLYING THE ADAPTED EMISQ MODEL

	Log4j		**JabRef**		**TV-Browser**		**RSSOwl**		**Checkstyle**	
	Avg4	Rank	Avg4	Rank	Avg4	Rank	Avg4	Rank	Avg4	Rank
FindBugs	4.393	4	4.393	4	4.464	3	4.857	2	5	1
PMD	3.253	3	2.425	5	3.126	4	3.678	2	4.264	1
SISSy	2.8	3	2.1	5	2.64	4	3.4	2	3.52	1
All Tools	3.309	3	2.661	5	3.206	4	3.794	2	4.164	1

Thus, the EMISQ quality model ranks *Checkstyle* as the best project and *JabRef* as the worst project. The *TV-Browser* project is on rank 4. This is the typical result we also saw from the expert evaluation presented in study 1. Given this result we can now compare the results of the EMISQ quality model with those of our IT-CISQ measures (as shown in Table 6 and Table 7).

2) Study Results and Discussion

As presented earlier, metrics M1, M2, M3 and M19 need some manual input. We tried in our experiment to automatically configure these rules, e.g. by automatically calculating a layering structure using the tool *Classycle* (see discussion in Section V.B. Nevertheless a review of the results indicate that the provided automatic configurations do not properly reflect the reality of the projects. We therefore concentrate for this discussion only on the results of *Variant A* of our IT-CISQ model. Correlation coefficients with one star denote a significant correlation assuming a level of significance of $\alpha = 0.05$. Entries with two stars are even significant at a level of significance of $\alpha = 0.01$. Again, as the sample size is small (5 projects), the correlation coefficient has to have a value of 0.878 ($\alpha = 0.05$) and 0.959 ($\alpha = 0.01$) to indicate a significant correlation. Table 11 shows the correlation results between the EMISQ quality model and the IT-CISQ Model *Variant A*.

TABLE 11: CORRELATION COEFFICIENT OF IT-CISQ VARIANT B VS. EMISQ QUALITY MODEL (MAINTAINABILITY ONLY)

	IT-CISQ Variant A
FindBugs	0.606
PMD	0.9*
SISSy	0.9*
All Tools	0.9*

Each row of Table 11 shows the correlation of the measures of a specific tool used in the EMISQ quality model and the IT-CISQ measures *Variant A*. Interestingly the correlation with FindBugs is not significant. In our EMISQ quality model we included 28 of the maintainability related

measures of the overall 300+ measures available in FindBugs. The strength of FindBugs is more on the detection of bug patterns and not on maintainability; the maintainability related measures of FindBugs are also oriented towards finding bugs, e.g., "*Invocation of equals with null argument"* as one example of such a measure. Therefore FindBugs does only find a comparably low number of problems. For our comparison with the IT-CISQ measures Variant A the result given for *All Tools* is more interesting, as this represents all measures of the EMISQ quality model for maintainability. The result indicates that our IT-CISQ model *Variant A* is capable of at least finding out the best and worst projects of a set of projects. Additionally, as shown in Table 6 (first row) the model also calculates a maintainability index, where the worst project gets a quality index of 1 and the best project gets a quality index of 4.33 with a maximum of 5 that is achievable. The variance of the results therefore is higher than with our EMISQ model (see

Table 10), where the worst project gets a value of 2.661 and the best project a value of 4.164. Regardless of the numeric results, as our EMISQ quality model for maintainability comprises substantially more measures, it gives more precise guidance about parts of the software that should be enhanced and which measures to concentrate on. This is important, as the EMISQ quality model has a broad range of measures with different criticality – from blockers and majors to measures that only have informational character.

3) Threats to Validity

One threat to the internal validity of the study is the fact that the original EMISQ quality model did not only concentrate on maintainability, but also on the quality criteria reliability and efficiency. As our EMISQ quality model contains explicit impacts from measures to quality criteria, it was easy for us to obtain only those measures that are relevant for maintainability.

Again, the ranking of a small number of projects (five in our case) is a threat to internal validity. Nevertheless, we defined high minimum correlation coefficients to accept a correlation as significant (e.g. 0.878 for $\alpha = 0.05$) and therefore address this internal threat explicitly.

The major threat to the external validity is the fact that the study only comprised five projects, meaning that the probability that the ranking of projects fits (using different quality models) is quite high. In total we analyzed 31 projects, i.e. 26 Java projects to calculate the benchmark distribution, and five projects whose quality was finally evaluated. So, we were heavily invested to address this external threat to validity in the study. We do not think that adding new open source projects to the analysis would considerably change the results, as we also know from our expert-based evaluation that *JabRef*, for example, is a bad project (from the point of view of internal code quality) while *Checkstyle* is an excellent one. The external validity of these studies could be enhanced by applying and comparing our quality models (EMISQ quality model for maintainability, IT-CISQ Model Variant A) to industrial strength projects. Unfortunately, we currently do not have access to a larger set of industrial projects.

VI. Conclusion And Future Work

In this paper we presented an operationalization of the maintainability part of the IT-CISQ quality model. In other words, we clarified some vague requirements and provided an implementation of these measures in our MUSE tool based on the specifications given in the IT-CISQ standard. This is an essential contribution, as we are not aware of any other scientific work or tools that already tried to provide automated quality evaluations based on IT-CISQ specifications. Furthermore, the MUSE library itself, combined with our benchmarking-based evaluation, is of practical relevance as the IT-CISQ standard can also be viewed as a refinement/operationalization of the ISO / IEC 25010 standard, which is well established in the industry and is considered to be important in order to get quality statements for software. Our work also showed that, at least for the quality attribute maintainability, the standard is formulated in a way that automation of the proposed measures is possible. However, some adaptations of the formulation of the standard could enhance the conciseness and clearness of the specifications. Implementing (parts of) the IT-CISQ standard therefore shows that the level of detail provided by the standard is sufficient enough to be able to automate the specified measures (at least for the quality criterion maintainability).

This is of course only one part of the story. The question remains whether the IT-CISQ standard specifies measures that can serve as reliable predictors for software quality. Therefore, and as another main contribution, we applied the IT-CISQ standard for the quality criterion maintainability on five Java open source projects and evaluated their quality using our benchmarking-based approach. We compared these results with the results of the LSV (published in [7] and [15]) where ten experts manually evaluated the quality (with emphasis on maintainability) of five open source Java projects. As one major result of this study, we can conclude that the IT-CISQ model for maintainability (or at least our interpretation and implementation of the IT-CISQ model) is significantly correlated with the results of the experts. The study therefore is a strong indicator that our implementation of the IT-CISQ measures for maintainability can reasonably be used to evaluate the maintenance quality of Java software products.

In previous work [6] we developed the operational EMISQ quality model that covers a number of quality criteria of the ISO / IEC 25010 standard. The quality criterion maintainability comprises for the programming language Java in our EMISQ quality model 165 measures, compared with 21 measures of the IT-CISQ quality model for maintainability. In the second study of this article we also presented that there is a significant correlation between the quality ranking of the five Java open source projects as calculated by our EMISQ quality model and our IT-CISQ quality model. Therefore there is a strong indicator that the IT-CISQ quality model is a good predictor for the quality criterion maintainability. Crucially, however, this does not

mean that our EMISQ quality model is useless; quite the contrary, as our EMISQ quality model provides much more maintainability related measures, and provides more enhancement hints about source code aspects that can improve the the maintainability of the code. Keeping this in mind, the IT-CISQ quality model for maintainability could be used for quality evaluations, but is only partially suitable for improvement programs, as the IT-CISQ model for maintainability covers only some aspects of maintainability, but does not reach the coverage and depth needed for thorough quality improvement activities. For such a purpose, our EMISQ quality model is better suited.

Our results indicate that the measures itself are effective to be used in practice to roughly evaluate the maintainability of (Java) software systems. Nevertheless, the lack of a comprehensive evaluation method makes it difficult to be directly applied in practice – here additional methods have to be developed and suggested by the OMG. In our studies we efficiently used our proved benchmarking based approach [7]. Our evaluation method described in [5] could of course also contribute to a more efficient application of the IT-CISQ quality model for industry.

The emphasis of this paper is on the validation of the maintainability related measures of the IT-CISQ model. In order to be able to carry out this validation, it was necessary to provide a tooling for the maintainability part of the IT-CISQ quality model. Though not the focus of this paper, the provided measures are innovative, as we are not aware of any other work in this field. Our MUSE-based approach (see Section III) therefore by itself is innovative and provides at least an efficient way to get the results for the IT-CISQ defined measures. The effectivity of our approach was shown by our validation.

Due to the positive results of this work, it would be interesting to automate other parts of the IT-CISQ standard, i.e., other quality criteria and to validate the applicability of these IT-CISQ defined measures. In order to strengthen the external validity of our study, further experiments with industrial or commercial software would be interesting, as our current results are based on open-source software only.

REFERENCES

[1] F. Deissenboeck, E. Juergens, K. Lochmann, S. Wagner, Software quality models: Purposes, usage scenarios and requirements, in: Proc. ICSE Workshop on Software Quality, 2009

[2] ISO/IEC 25010:2011, Systems and software engineering – systems and software quality requirements and evaluation (SQuaRE), 2011

[3] S. Wagner, K. Lochmann, S. Winter, A. Goeb, M. Klaes, Quality models in practice: A preliminary analysis, in: Proc. 3rd, International Symposium on Empirical Software Engineering and Measurement, IEEE Computer Society, 2009

[4] ISO/IEC 25040:2011, Systems and software engineering - systems and software quality requirements and evaluation (SQuaRE) - evaluation process, 2011.

[5] S. Wagner, K. Lochmann, L. Heinemann, M. Klaes, A. Trendowicz, R. Ploesch, A. Seidl, A. Goeb, J. Streit, The Quamoco product quality modelling and assessment approach, in: Proc. 34th International Conference on Software Engineering, IEEE Computer Society, 2012

[6] R. Ploesch, H. Gruber, A. Hentschel, C. Koerner, G. Pomberger, S. Schiffer, M. Saft, S. Storck, The EMISQ method and its tool support - Expert-based evaluation of internal software quality, Innovations in Systems and Software Engineering 4 (2008) 3-15.

[7] H. Gruber, R. Ploesch, M. Saft, On the Validity of Benchmarking for Evaluating Code Quality, Proceedings of the Joined International Conferences on Software Measurement IWSM/MetriKon/Mensura 2010, November 10-12, Stuttgart, Germany, Shaker Verlag. Aachen 2010

[8] H. Gruber, R. Ploesch, S. Schiffer, A. Hentschel, Calculating Software Maintenance Risk - A Practical Approach, Proceedings of 11th IASTED International Conference on Software Engineering (IASTED SE 2012), June 18-20, Cretan, Greece, 2012

[9] CISQ Specifications for Automated Quality Characteristic Measures, CISQ Technical Work Groups for Reliability, Performance Efficiency, Security, Maintainability, CISQ-TR-2012-01, OMG, 2012

[10] F. Simon, O. Seng, T. Mohaupt, Code-Quality-Management - Technische Qualität industrieller Softwaresysteme transparent und vergleichbar gemacht; dpunkt Verlag, Heidelberg, 2006

[11] H. Gruber, C. Körner, R. Ploesch, G. Pomberger, S. Schiffer, Benchmarking-oriented Analysis of Source Code Quality – Experiences with the QBench Approach, Proceedings of the International Conference on Software Engineering (IASTED SE 2008), 2008

[12] H. Gruber, R. Ploesch, M. Saft, On the Validity of Benchmarking for Evaluating Code Quality, Proceedings of the Joined International Conferences on Software Measurement IWSM/MetriKon/Mensura 2010, November 10-12, Stuttgart, Germany, Shaker Verlag. Aachen 2010

[13] S.R. Chidamber, C.F. Kemerer, A Metrics Suite for Object-Oriented Design, IEEE Trans. Software Engineering, vol. 20, no. 6, pp. 476-493, June 1994.

[14] T. McCabe. A Complexity Measure. In: IEEE Transactions on Software Engineering, 2.4 (1976), S. 308–320.

[15] H. Gruber, A. Hentschel, R. Ploesch, Experiment for Comparing the Automatically Assessed Source Code Quality with Experts' Opinions. In: Proceedings of the SQMB 2009 Workshop, held with the SE 2009 Conference. 2009.

[16] P. C. Fishburn, Condorcet Social Choice Functions. In: SIAM Journal on Applied Mathematics 33.3 (1977), S. 469–489.

[17] M. Schulze, A new monotonic and clone-independent single-winner election method. In: Voting matters 17.1 (2003), S. 9–19.

[18] F.-J. Elmer, Classycle: Analysing Tools for Java Class and Package Dependencies. 2012. url: http://classycle.sourceforge.net/

[19] Versant Corporation. db4o - Java & .NET Object Database. 2014. url: http://www.db4o.com/

[20] Apache Software Foundation. Apache Commons IO. 2014. url: http://commons.apache.org/proper/commons-io/